

# GROUT: A 1-DIMENSIONAL SUBSTITUTION TILING SPACE PROGRAM

SCOTT BALCHIN & DAN RUST

**ABSTRACT.** We introduce a GUI fronted program that can compute combinatorial properties and topological invariants of recognisable and primitive symbolic substitutions on finite alphabets and their associated tiling spaces. We introduce theory from the study of aperiodic 1-dimensional tilings along with pseudocode highlighting the algorithms that we have implemented into the GUI. Grout is written using C++ and its standard library.

## Keywords

Tiling spaces, Symbolic dynamics, Čech cohomology, Substitutions.

## SUPPLEMENTARY RESOURCES

Grout is available to download for Windows and Mac OSX along with the supporting documentation at the following URL

[www2.le.ac.uk/departments/mathematics/extranet/staff-material/staff-profiles/scott-balchin](http://www2.le.ac.uk/departments/mathematics/extranet/staff-material/staff-profiles/scott-balchin)

## 1. INTRODUCTION TO TILING SPACES

The study of aperiodic tilings of the plane has a rich history which emerged from the worlds of computer science and mathematical logic when Berger proved the undecidability of the domino problem in the 1960s [6]. It is now also a topic of general interest to the study of dynamical systems, topology, Diophantine approximation, ergodic theory, computer graphics, mathematical physics and even virology [9, 25, 7, 3, 20, 5, 27]. Once it was discovered that sets of tiles exist which can only tile the plane aperiodically, a flurry of new discoveries quickly followed, culminating in the celebrated discovery of the famous *Penrose tilings* [22] and in the discovery of quasicrystals [26] for which Schectman was awarded the Nobel prize in Chemistry in 2011.

The seminal paper of Anderson and Putnam [2] lead to a new algebraic topological approach to aperiodic tilings associated to a particular method of generating tilings of the plane known as the *substitution method*. It is this method, but restricted to a 1-dimensional analogue, which we address here.

It is hoped that the use of this program will make testing conjectures in tiling theory and symbolic substitutional dynamics more efficient, as well as allowing for the confirmation of hand calculations and comparing different methods of calculation (especially methods of calculating cohomology). Analysis of large data sets which can be potentially generated by the Grout source code, and the recognition of underlying patterns in the data may also aid to further the theory.

The GUI front end for Grout is powered by Qt[1]. Grout has been designed with user experience in mind and includes many ease-of-use properties such as the ability to save and load examples, and convenient methods of sharing examples with other users via short strings that encode a substitution. There is also an option to export all of the data that has been calculated to a pre formatted  $\text{\LaTeX}$  file including all the TikZ code for the considered complexes. This should be useful for those needing to typeset such diagrams in the future by fully automating the generation of diagrams in TikZ.

---

The University of Leicester, University Road, Leicester, LE1 7RH, United Kingdom.

In Section 1.1 we introduce basic notions of tiling spaces associated to substitutions on finite alphabets. In Section 2 we introduce the relevant tiling theory along with pseudocode for most of the non-trivial components of Grout that have been implemented. Section 3 will cover specifically those methods implemented to compute cohomology for tiling spaces. Throughout, we give instances of these methods being applied to a well-known example substitution. In Section 4 we give a wide range of other examples of calculations from Grout.

**Acknowledgements.** The authors would like to thank Fabien Durand for his helpful advice in piecing together a robust recognisability algorithm, and Etienne Pillin for helpful comments with regards to the coding. We would also like to thank Alex Clark, Greg Maloney and Jamie Walton for helpful suggestions during the development of Grout and for testing early versions of the program.

## 1.1. Background.

**Definition 1.1.** Let  $\mathcal{A} = \{a^1, \dots, a^l\}$  be a finite alphabet on  $l$  symbols, and for all positive integers  $n$  let  $\mathcal{A}^n$  be the set of length  $n$  words in  $\mathcal{A}$ . Denote the union of these as  $\mathcal{A}^+ = \bigcup_{n \geq 1} \mathcal{A}^n$ . If, further, the empty word  $\epsilon$  is included, we denote the union  $\mathcal{A}^+ \cup \{\epsilon\}$  by  $\mathcal{A}^*$ . A *substitution*  $\phi$  on  $\mathcal{A}$  is a function  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$  which assigns to each letter  $a$  in  $\mathcal{A}$  a non-empty word  $\phi(a)$  whose letters are elements in  $\mathcal{A}$ . We extend  $\phi$  to a function  $\phi: \mathcal{A}^+ \rightarrow \mathcal{A}^+$  by concatenation; given a word  $w = w_1 \dots w_n \in \mathcal{A}^n$ , we set  $\phi(w) = \phi(w_1) \dots \phi(w_n)$ . In this way, we can consider finite iterates of the substitution  $\phi^n$  acting on  $\mathcal{A}^+$ .

**Definition 1.2.** Let  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$  be a substitution. We say a word  $w \in \mathcal{A}^*$  is *admitted* by the substitution  $\phi$  if there exists a letter  $a \in \mathcal{A}$  and a natural number  $n \geq 0$  such that  $w$  is a subword of  $\phi^n(a)$  and denote by  $\mathcal{L}^n \subset \mathcal{A}^n$  the set of all words of length  $n$  which are admitted by  $\phi$ . Our convention is that the empty word  $\epsilon$  is admitted by all substitutions. We form the *language* of  $\phi$  by taking the set of all admitted words  $\mathcal{L} = \bigcup_{n \geq 0} \mathcal{L}^n$ .

We say a bi-infinite sequence  $s \in \mathcal{A}^{\mathbb{Z}}$  is *admitted* by  $\phi$  if every subword of  $s$  is admitted by  $\phi$  and denote by  $X_\phi$  the set of all bi-infinite sequences admitted by  $\phi$ . The symbol  $s_i$  denotes the label assigned to the  $i$ th component of the sequence  $s$ . The set  $X_\phi$  has a natural (metric) topology inherited from the product topology on  $\mathcal{A}^{\mathbb{Z}}$  and a natural shift map  $\sigma: X_\phi \rightarrow X_\phi$  given by  $\sigma(s)_i = s_{i+1}$ . We call the pair  $(X_\phi, \sigma)$  the *subshift* associated to  $\phi$  and we will often abbreviate the pair to just  $X_\phi$  when the context is clear.

We say  $\phi$  is a *periodic* substitution if  $X_\phi$  is finite, and say  $\phi$  is aperiodic otherwise. If  $\phi$  is *aperiodic* and has a property which will be introduced later known as *primitivity*, then  $X_\phi$  is a Cantor set (in particular  $X_\phi$  is non-empty) and  $\sigma$  is a minimal action on  $X_\phi$  - that is, the only closed shift-invariant subsets of  $X_\phi$  are the empty set  $\emptyset$  and the subshift itself  $X_\phi$ . Equivalently, the orbit of every point under  $\sigma$  is dense in  $X_\phi$ .

**Definition 1.3.** Let  $\phi$  be a substitution on the alphabet  $\mathcal{A}$  with associated subshift  $X_\phi$ . The *tiling space* associated to  $\phi$  is the quotient space

$$\Omega_\phi = (X_\phi \times [0, 1]) / \sim$$

where  $\sim$  is generated by the relation  $(s, 0) \sim (\sigma(s), 1)$ .

One should imagine the point  $(s, t) \in \Omega_\phi$  as being a partition or tiling of the real line  $\mathbb{R}$  into labelled unit-length intervals (called tiles), where the labels are determined by the letters appearing in  $s$  and the origin of  $\mathbb{R}$  is shifted a distance  $t$  to the right from the left of the tile labelled by the letter  $s_0$ . Two tilings  $T$  and  $T'$  in  $\Omega_\phi$  are considered  $\epsilon$ -close in this topology if, after a translate by a distance at most  $\epsilon$ , the tiles around the origin in  $T' - \epsilon$  within a ball of radius  $1/\epsilon$  lie exactly on top of the tiles around the origin in  $T$  within a ball of the same radius and share the same labels.

If  $\phi$  is primitive and aperiodic, then  $\Omega_\phi$  is a compact connected metric space which fibers over the circle with Cantor set fibers. The natural translation  $T \mapsto T + t$  for  $t \in \mathbb{R}$  equips  $\Omega_\phi$  with a continuous  $\mathbb{R}$  action which is minimal as long as  $\phi$  is primitive. In this respect, tiling spaces are closely related to the more well-known spaces, the solenoids.

Associated to any topological space  $X$  is a collection of groups  $\check{H}^*(X)$  called the *Čech cohomology* of  $X$ . We refer the reader to [8] for an introduction to Čech cohomology. Čech cohomology is an important topological invariant of tiling spaces and it is of general interest to be able to calculate and study these groups. Grout implements three different methods for calculating the cohomology of tiling spaces associated to symbolic substitutions on finite alphabets.

- (1) The method of Barge-Diamond complexes as introduced in [4]
- (2) The method of Anderson-Putnam complexes as introduced in [2]
- (3) The method of forming an equivalent *left proper* substitution as outlined in [13]

All three outputs are algebraically equivalent – that is, they represent isomorphic groups – but it is not always obvious that this is the case given the presentations. This disparity between presentations of results for the equivalent methods was one of the major motivating factors for developing Grout. These cohomologies are extremely laborious to calculate by hand for large alphabets unless special criteria are met.

## 2. GROUT AND ITS FUNCTIONS

**2.1. Substitution Structure.** We begin by outlining how we encode a substitution rule into Grout and how we implement the substitution rule. In general, we have done most of the implementation by string manipulation methods.

We use a class *sub* which has as its element a vector of strings. We always assume that our alphabet is ordered  $a, b, c, \dots$ . The first entry of a *sub* class vector is  $\phi(a)$ , the second is  $\phi(b)$  and so on. To validate the input we check that the number of unique characters appearing in all of the  $\phi(x)$  is equal to the length of the alphabet, which is the length of the vector. The GUI also employs the use of regular expressions to prevent illegal characters from being entered.

**Example 2.1** (Fibonacci Substitution). The Fibonacci tiling is given by a substitution rule on the alphabet  $\mathcal{A} = \{a, b\}$  and is defined as

$$\phi: \begin{cases} a \mapsto b \\ b \mapsto ba \end{cases}$$

The Fibonacci substitution is our main example used throughout the paper. See section 4 for a selection of outputs for other common examples of substitutions.

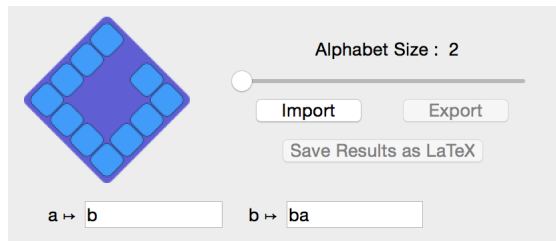


FIGURE 1. The Fibonacci substitution entered into Grout

Next we implement a way to perform an iteration of  $\phi$  on a string. We do not include any checks to validate that the string can be iterated on, as all strings that will be passed to this function will be created by the program itself, and therefore valid.

---

**Algorithm 1** Substitution functions

---

```
1: function iterate(string rhs)
2:   result = empty string
3:   for each character x in rhs do
4:     append  $\phi(x)$  to result
5:   output result
```

---

## 2.2. Substitution Matrices and their Properties.

**Definition 2.2.** Let  $\mathcal{A} = \{a^1, \dots, a^l\}$  be an alphabet with a substitution  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$ , then  $\phi$  has an associated *substitution matrix*  $M_\phi$  of dimension  $l \times l$  given by setting  $m_{ij}$  to be the number of times that the letter  $a^i$  appears in  $\phi(a^j)$ .

We will not give the algorithm of constructing the substitution matrix, the definition can be taken as a pseudo-algorithm. We implement a square matrix class to work with the substitution matrix. The first property that we will be checking for the substitution matrix is primitivity.

**Definition 2.3.** A substitution  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$  is called *primitive* if there exists a positive natural number  $p$  such that the matrix  $M_\phi^p$  has strictly positive entries. Such a matrix  $M$  is also called *primitive*. This condition is equivalent to asking that there is a positive natural number  $p$  such that for all  $a, a' \in \mathcal{A}$  the letter  $a'$  appears in the word  $\phi^p(a)$ .

We will use the first definition to check the primitivity of our substitutions. To check this condition on  $M_\phi$ , we count the number of zeros in the matrix and if the number of zeros reaches 0 then we can conclude that the substitution is primitive. If not, then square the matrix and recount the number of zeros. If the number of zeros does not change then we can conclude that the substitution is not primitive. This means that this check always halts.

---

**Algorithm 2** Primitivity check

---

```
1: function primitive
2:   matrix = substitution matrix of  $\phi$ 
3:   while true do {
4:     a = number of zeros in matrix
5:     matrix = matrix  $\times$  matrix
6:     b = number of zeros in matrix
7:     if a=b and a!=0
8:       output false
9:     if b=0
10:      output true
11:   }
```

---

We will be checking primitivity for all substitutions before we do calculations on them as if the substitution is not primitive many of the methods will not work, or will return false positive results. Grout will always display whether a given substitution is primitive or not, it can also output the substitution matrix if asked to do so.

The next thing that we can do with the substitution matrix is give the tile frequencies and tile lengths of the substitution. This requires us to compute the eigenvalues of the matrix. We have implemented the QR method for computing the eigenvalues (for example see [17]). This gives us approximations to the real eigenvalues, and for the complex ones we simply give the conjugate pairs by their absolute values, and we give the results to two decimal places. The eigenvalues of a substitution matrix may be printed out by ticking the eigenvalues box.

**Proposition 2.4** (Perron-Frobenius). *Let  $M$  be a primitive matrix.*

- i *There is a positive real number  $\lambda_{PF}$ , called the Perron-Frobenius eigenvalue, such that  $\lambda_{PF}$  is a simple eigenvalue of  $M$  and any other eigenvalue  $\lambda$  is such that  $|\lambda| < \lambda_{PF}$ .*
- ii *There exist left and right eigenvectors, called the left and right Perron-Frobenius eigenvectors,  $\mathbf{l}_{PF}$  and  $\mathbf{r}_{PF}$  associated to  $\lambda_{PF}$  whose entries are all positive and which are unique up to scaling.*

Given the above theorem, it is natural to ask what information is contained in the PF eigenvalue and eigenvectors of  $M_\phi$  for a primitive substitution  $\phi$ .

If we were to assign a length to the tiles labelled by each letter, then we would hope for such a length assignment to behave well with the given substitution. The left PF eigenvector offers a natural choice of length assignments. If we assign to the letter  $a^i$  the length  $(\mathbf{l}_{PF})_i$ , the  $i$ th component of the left PF eigenvector, then we can replace our combinatorial substitution by a geometric substitution. This geometric substitution expands the tile with label  $a^i$  by a factor of  $\lambda_{PF}$  and then partitions this new interval into tiles of lengths and labels given according to the combinatorial substitution. In order to give a unique output, Grout normalises the left PF eigenvector so that the smallest entry is 1.

The information contained in the right PF eigenvector is also useful. The right PF eigenvector, once normalised so that the sum of the entries is 1, gives the relative frequencies of each of the letters appearing in any particular bi-infinite sequence which is admitted by  $\phi$ . That is, if  $|w|$  is the length of the word  $w$ ,  $|w|_i$  is the number of times the letter  $a^i$  appears in the word  $w$ , and letting  $s_{[-k,k]} = s_{-k} \dots s_{-1} s_0 s_1 \dots s_k$ , then

$$\lim_{k \rightarrow \infty} |s_{[-k,k]}| / |s_{[-k,k]}|_i = (\mathbf{r}_{PF})_i$$

for any  $s \in X_\phi$ .



FIGURE 2. The results for the matrix calculations for the Fibonacci substitutions

**2.3. Enumerating  $n$ -Letter Words.** Now that we have introduced the basic structure of the substitutions, and discussed the problem of primitivity and other matrix related calculations, we will discuss our first main function in Grout.

**Definition 2.5.** Given a substitution  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$ , we define the *complexity function* at  $n$  to be the number of unique  $n$ -letter words admitted by  $\phi$ . We denote this function by  $p_\phi(n)$ , and so  $p_\phi(n) = |\mathcal{L}_n|$ .

The complexity function of a tiling is a useful invariant [19]. One is usually interested in either a deterministic formula for  $p_\phi$  or information about the growth rate of  $p_\phi$  such as polynomial degree; Grout can be used to at least give circumstantial evidence for these, though has no means of calculating either (this appears to be a very difficult problem in general).

Of particular interest are the number of 2 and 3 letter words, as we will be using them later to compute cohomology. Our function will not only enumerate the number of  $n$ -letter words, but will also print out these words if required. The algorithm uses C++ sets as a data structure to store the  $n$ -letter words as it is automatically ordered and does not allow repetitions which leads to fast computation. We start by generating a length  $m$  admitted seed word  $w$  such that  $m \geq n$ , and count all unique  $n$ -letter words appearing as subwords of the seed. We then apply  $\phi$  to the seed and add all new  $n$ -letter words to the result. At each stage we count the size before and after adding the new words. If the size does not change we can stop, as no new  $n$ -letter words will be generated after a step without any new  $n$ -letter words. It follows that the value  $p_\phi(n)$  is computable in finite time for any fixed  $n \geq 1$ .

---

**Algorithm 3** Finding all  $n$ -letter words

---

```

1: function nlw(int n)
2:   result = empty ordered set
3:   seed = 'a'
4:   while seed length < n do
5:     seed = iterate(seed)
6:
7:   difference = 1
8:   while difference != 0 do {
9:     a = cardinality of result
10:    seed = iterate(seed)
11:    for each  $n$  length word  $w$  in seed do
12:      append  $w$  to result
13:    b = cardinality of result
14:    difference = b-a
15:  }
16: output result

```

---

**Example 2.6.** It is well known that the complexity for the Fibonacci substitution satisfies  $p_\phi(n) = n + 1$ , and we can verify this for any value of  $n$  by computing its complexity in Grout.

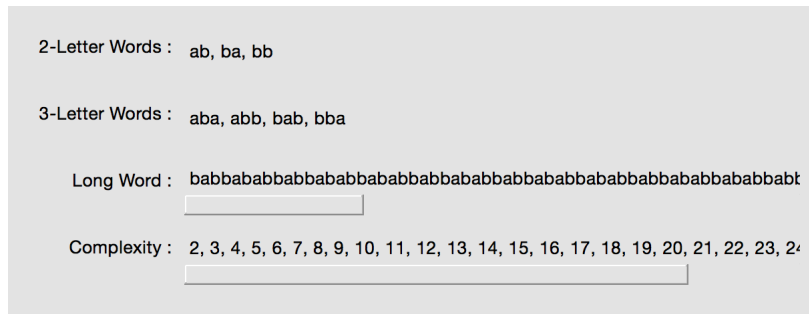


FIGURE 3. The words results for the Fibonacci substitution displayed in Grout

**2.4. Barge-Diamond and Anderson-Putnam Complexes.** Grout has the ability to output two simplicial complexes as PDFs (provided that the user has PDFLaTeX installed). The first of these is the *Barge-Diamond complex* [4], which will be the key tool used in one of the methods of computing the Čech cohomology of the tiling space.

**Definition 2.7.** Let  $\mathcal{A} = \{a^1, \dots, a^l\}$  be an alphabet with a substitution  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$ , then we construct the *Barge-Diamond complex* of  $\phi$  as follows. We have two vertices for each  $a^i$ , an *in* node  $v_i^+$  and an *out* node  $v_i^-$ . We draw an edge from  $v_i^+$  to  $v_i^-$  for all  $i$ . Then for all two letter words  $a^i a^j \in \mathcal{L}^2$  admitted by  $\phi$ , we draw an edge from  $v_i^-$  to  $v_j^+$ .

**Example 2.8.** As we have seen previously, the only two letter words admitted by the Fibonacci substitution are  $ab, ba$  and  $bb$ , this gives us the following Barge-Diamond complex output in Grout.

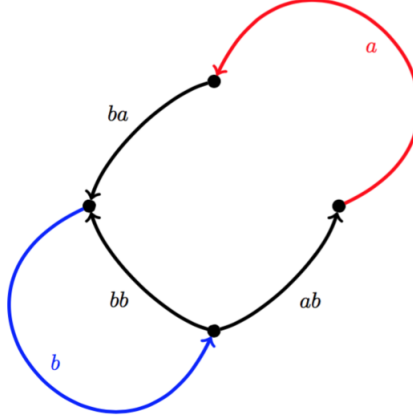


FIGURE 4. The Barge-Diamond complex for the Fibonacci substitution.

The other complex that we consider for a substitution is (a variant of) the *collared Anderson-Putnam complex* [2] which will again be the key tool used in one of the methods of computing Čech cohomology of the tiling space. For brevity we will often shorten this to the *AP complex*. This particular definition is based on what Gähler and Maloney call the *Modified Anderson-Putnam complex* in [16]. The AP complex is constructed in a similar fashion to the Barge-Diamond complex, but makes use of both the two and three letter words.

**Definition 2.9.** Let  $\mathcal{A} = \{a^1, \dots, a^l\}$  be an alphabet with a substitution  $\phi: \mathcal{A} \rightarrow \mathcal{A}^+$ , then we construct the *Anderson-Putnam complex* of  $\phi$  as follows. We have a vertex  $v_{ij}$  for each two letter word  $a^i a^j \in \mathcal{L}^2$  admitted by  $\phi$ . We draw an edge from  $v_{ij}$  to  $v_{jk}$  if and only if the three letter word  $a^i a^j a^k$  is admitted by  $\phi$ .

**Remark 2.10.** One should note that this modified AP complex is slightly different to the definition originally introduced by Anderson and Putnam. In particular, the original definition distinguishes between different occurrences of a two letter word  $a^i a^j$  if the occurrences of three letter words containing as a subword  $a^i a^j$  do not overlap on some admitted four letter word. For example, if the language of a substitution included the two letter word  $ab$ , the three letter words  $xab, yab, abw, abz$ , and the four letter words  $xabw, yabz$  but the words  $xabz$  and  $yabw$  did not belong to  $\mathcal{L}$ , then the original definition of the AP complex would have two instances of vertices with the label  $ab$ , say  $(ab)_1$  and  $(ab)_2$ . In our definition, these vertices are identified, so that  $(ab)_1 \sim (ab)_2 \sim ab$ . An example of such a substitution is given by  $\phi: a \mapsto bc, b \mapsto baab, c \mapsto caac$  where we label exactly one vertex with the label  $aa$ , but the original definition would require we include two distinct vertices labelled  $(aa)_1$  and  $(aa)_2$ .

In our discussion of cohomology calculated via AP complexes in Section 3.2, we use this version of the AP complex to describe the performed calculations, and Grout implements this particular method. It would have been possible to use the original definition, or one of the many variant AP complexes that have been defined in the literature. There are at least three such variants discussed in [16], of varying complexities and situations in which they can be used.

**Example 2.11.**

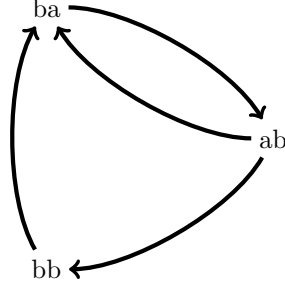


FIGURE 5. The Modified Anderson-Putnam complex for the Fibonacci substitution

## 2.5. Recognisability.

**Definition 2.12.** Let  $\phi$  be a substitution on the alphabet  $\mathcal{A}$ . We say  $\phi$  is *recognisable* if for every bi-infinite sequence  $s \in X_\phi$  admitted by  $\phi$  there is a unique way of decomposing  $s$  into words of the form  $\phi(a)$  for  $a \in \mathcal{A}$ . That is, up to a finite shift, there exists a unique bi-infinite sequence  $\dots a_{-1}a_0a_1\dots$  such that  $s = \dots \phi(a_{-1})\phi(a_0)\phi(a_1)\dots$  and so we can *recognise* which substituted letter each letter in  $s$  has come from.

Equivalently, we say  $\phi$  is recognisable if there exists a natural number  $K \geq 1$  such that for all admitted words  $w \in \mathcal{L}$  with  $|w| > 2K$ , there exist unique words  $x, y$  of length  $|x|, |y| \leq K$  and a unique admitted word  $u \in \mathcal{L}$  such that  $w = x\phi(u)y$ .

Recognisability is an important property of a tiling as many of the tools used to study the topology of the associated tiling spaces rely on recognisability as a hypothesis, much like primitivity. Recognisability of a primitive substitution is equivalent to aperiodicity of the subshift  $X_\phi$  [21], and we make use of this result to decide recognisability. The algorithm designed to determine if a given substitution is recognisable relies on finding a fixed letter and return words to that fixed letter.

**Definition 2.13.** Given a substitution  $\phi$  on an alphabet  $\mathcal{A}$ , the letter  $a$  is said to be *fixed* (on the left) of *order*  $k$  if there exists some integer  $k$  such that  $\phi^k(a) = au$  for some word  $u$ . Every substitution has at least one fixed letter and the value of  $k$  for such a letter is bounded by the size of the alphabet.

**Definition 2.14.** Given a fixed letter  $a$ , a *return word* to  $a$  is a word  $v$  such that  $v = au$  for some (possibly empty) word  $u \in (\mathcal{A} \setminus \{a\})^*$ , and  $va$  is an admitted word of the substitution.

If  $\phi$  is primitive then, due to the primitivity of the substitution, the set of return words to any letter is finite. This is a consequence of the *linear recurrence* of the subshift  $X_\phi$ , shown by Damanik and Lenz [11]. We omit a definition of linear recurrence here.

We will use these return words to determine whether a substitution is recognisable or not. The following proposition appears in [18].



---

**Algorithm 4** Finding all return words to fixed letter  $f$ 

---

```
1: function returnwords(character  $f$ )
2:   result = empty ordered set
3:   length = 2
4:   while new return words are being added do {
5:     nwords = nlw(length)
6:     for all words  $w$  in nwords do {
7:       if last character of  $w$  = first character of  $w = f$  and  $w$  has no other  $f$  appearing do
8:         append  $w$  to result
9:     }
10:    length = length + 1
11:  }
12:  output result
```

---

**Proposition 2.15.** *Let  $\phi$  be a primitive substitution on  $\mathcal{A}$  and let  $a$  be a fixed letter. Let  $\mathcal{R}$  be the set of all return words to  $a$ . So  $\mathcal{R} = \{v \mid v = au, aua \in \mathcal{L}, u \in (\mathcal{A} \setminus \{a\})^*\}$ . The substitution  $\phi$  is not recognisable if and only if, for all  $v, v' \in \mathcal{R}$ , there exists a  $p \geq 1$  such that  $\phi^p(vv') = \phi^p(v'v)$ .*

As  $\mathcal{R}$  is finite, and together with the next proposition which appears in [10] and [15], this gives us a finite deterministic check for recognisability.

**Proposition 2.16.** *Let  $\phi$  be a substitution on  $\mathcal{A}$  and let  $|\mathcal{A}| = n$ . For words  $u, w \in \mathcal{A}^+$ , there exists a  $p \geq 1$  such that  $\phi^p(u) = \phi^p(w)$  if and only if  $\phi^n(u) = \phi^n(w)$ .*

That is, if some iterated substitution of  $u$  and  $v$  are ever equal, then their iterates must become equal at least by the  $n$ th iteration of the substitution, where  $n$  is the size of the alphabet. In the algorithm,  $k$  is taken to be the  $k$  from the definition of the fixed letter, and  $n$  is the size of the alphabet.

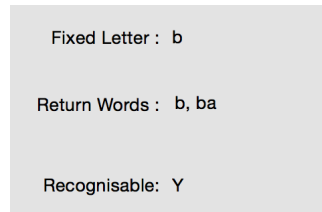
---

**Algorithm 5** Recognisability check

---

```
1: function recognisable
2:   rwords = returnwords( $f$ )
3:   for each word  $w$  in rwords do
4:     for each word  $v \neq w$  in rwords do
5:       if ( $\phi^{k \times n}(w + v) = \phi^{k \times n}(v + w)$ )
6:         output true
7:   output false
```

---



Fixed Letter : b

Return Words : b, ba

Recognisable: Y

FIGURE 6. The recognisability results for the Fibonacci substitution

### 3. COHOMOLGY OF TILING SPACES IN GROUT

**3.1. Via Barge-Diamond.** Let  $\phi$  be a primitive, recognisable substitution on the alphabet  $\mathcal{A}$ . Let  $G$  be the Barge-Diamond complex of  $\phi$  and let  $S$  be the subcomplex of  $G$  formed by all those edges labelled with two letter admitted words  $a^i a^j$ .

Let  $\tilde{\phi}: S \rightarrow S$  be a graph morphism defined in the following way on vertices. Let  $l(i)$  and  $r(i)$  be such that  $\phi(a^i) = a^{l(i)} u a^{r(i)}$  and define  $\tilde{\phi}(v_i^+) = v_{l(i)}^+$  and  $\tilde{\phi}(v_i^-) = v_{r(i)}^-$ . Note that if  $a^i a^j$  is admitted by  $\phi$ , then  $a^{r(i)} a^{l(j)}$  is also admitted by  $\phi$ , and so  $\tilde{\phi}$  is a well defined graph morphism on  $S$ . As  $\tilde{\phi}(S) \subset S$ , we can define the eventual range  $ER = \bigcap_{m \geq 0} \tilde{\phi}^m(S)$  (which stabilises after finitely many substitutions).

For this method of computation we make use of the following result attributed to Barge and Diamond [4].

**Proposition 3.1.** *There is a short exact sequence*

$$0 \rightarrow \tilde{H}^0(ER) \rightarrow \varinjlim M_\phi^T \rightarrow \check{H}^1(\Omega_\phi) \rightarrow H^1(ER) \rightarrow 0.$$

The eventual range  $ER$  is a (possibly disconnected) graph, and so  $\tilde{H}^0(ER)$  and  $H^1(ER)$  are finitely generated free abelian groups of some ranks  $k$  and  $l$  respectively. Hence we have

**Corollary 3.2.**

$$\check{H}^1(\Omega_\phi) \cong \varinjlim M_\phi^T / \mathbb{Z}^k \oplus \mathbb{Z}^l.$$

Grout displays the Čech cohomology using the Barge-Diamond method in the above form.

These results fail in general if  $\phi$  is not primitive or recognisable, and so Grout only performs this calculation after checking these two conditions.

The only involved part of this calculation is finding the eventual range of the Barge-Diamond complex. After we have this we can simply find the number of connected components and use the Euler characteristic to find the reduced cohomology in rank zero and one. Therefore, we now give the algorithm that we use to find the eventual range. We denote by  $w[0]$  the first letter of the word  $ab$  and  $w[1]$  the second letter of  $ab$ .

---

**Algorithm 6** Eventual range of the Barge-Diamond complex

---

```

1: function eventual_range
2:   twoletter = nlw(2)
3:   difference = 1
4:   while difference != 0 do {
5:     temp = empty ordered set
6:     for each word  $w$  in twoletter do
7:       append last character of iterate( $w[0]$ ) + first character of iterate( $w[1]$ ) to temp
8:     difference = cardinality of twoletter - cardinality of temp
9:     twoletter = temp
10:  }
11: output twoletter

```

---

**3.2. Via Anderson-Putnam.** Let  $\phi$  be a primitive, recognisable substitution on the alphabet  $\mathcal{A}$ . Let  $K$  be the Anderson-Putnam complex of  $\phi$ . Anderson and Putnam showed in [2] that the Čech cohomology of  $\Omega_\phi$  is determined by the direct limit of an induced map acting on the cohomology of a CW complex. Using the modified AP complex  $K$ , Gähler and Maloney showed that this complex, as defined in Section 2.4, can be used in place of the originally defined AP complex. We define the map acting on the AP complex  $K$  in the following way.

Again let  $l(i)$  and  $r(i)$  be such that  $\phi(a^i) = a^{l(i)}ua^{r(i)}$ . Let  $E$  be an edge with label  $a^i a^j a^k$  and define  $L = |\phi(a^j)|$ . Suppose  $\phi(a^j) = a_1 a_2 \dots a_L$ . Define a continuous map called the *collared substitution*  $\tilde{\phi}: K \rightarrow K$  by mapping the edge  $E$  to the ordered collection of edges with labels

$$[a^{r(i)} a_1 a_2][a_1 a_2 a_3] \cdots [a_{L-2} a_{L-1} a_L][a_{L-1} a_L a^{l(k)}]$$

in an orientation preserving way and at normalised speed. This map is well defined and continuous, hence an induced map  $\tilde{\phi}^*: H^1(K) \rightarrow H^1(K)$  on cohomology exists. We use the following result from [16] (Or [2] if  $K$  is replaced with the original definition of the AP complex).

**Proposition 3.3.**

$$\check{H}^1(\Omega_\phi) \cong \varinjlim (H^1(K), \tilde{\phi}^*)$$

Let  $R = \text{rk} H^1(K)$ , the rank of the cohomology of  $K$ . Grout finds an explicit generating set of cocycles for the cohomology of  $K$  and then outputs the induced map  $\tilde{\phi}^*$  as the associated  $R \times R$ -matrix  $M_{AP}$ , which should be interpreted as acting on  $\mathbb{Z}^R$  with respect to this generating set. So  $\check{H}^1(\Omega_\phi) \cong \varinjlim M_{AP}$ .

The algorithm for this computation begins by constructing the boundary matrix for the AP-complex. To do this we take all of the admitted three letter words  $abc$  and we use the convention that the boundary of this edge is  $bc - ab$ . Using this, we construct the associated  $m \times n$  boundary matrix  $B$  where  $m$  is the number of two letter words and  $n$  is the number of three letter words. We then use standard methods from linear algebra to find a maximal set of linearly independent  $n$  dimensional vectors  $g$  such that  $Bg = 0$ , searching over the set of all vectors of 0s and 1s. By construction, this set generates the kernel of the boundary map inside the simplicial 1-chain group of  $K$ . This gives us a generating set of cycle vectors for the first homology of  $K$ .

We then apply the collared substitution to each of these generating vectors, giving us a new set of image vectors. Using Gaussian elimination, we find the coordinates of these image vectors in terms of the generating vectors. This induced map on homology can be represented as a square matrix. The transpose of this matrix  $M_{AP}$  then represents the induced map on cohomology, and  $M_{AP}$  is the output for the cohomology calculation via the Anderson-Putnam method. It should be noted that this algorithm is not efficient in the case where the substitution has many three letter words, as the dimension  $m$  of the 1-chain complex is the dominant limiting factor when finding linearly independent generating cycles. The time complexity increases exponentially with respect to  $m$ .

**3.3. Via Properisation.** For this method of computation we make use of a technique involving return words, as outlined in [13], for replacing a primitive substitution with an equivalent *pre-left proper* primitive substitution. One may then use the fact that if  $\phi$  is a recognisable pre-left proper primitive substitution, then  $\check{H}^1(\Omega_\phi) \cong \varinjlim M_\phi^T$  (See [25]).

We begin by defining what it means to be proper.

**Definition 3.4.**

A substitution is *left proper* if there exists a letter  $a \in \mathcal{A}$  such that the leftmost letter of  $\phi(b)$  is  $a$  for all  $b \in \mathcal{A}$ . That is,  $\phi(b) = aw_b$  for some  $w_b \in \mathcal{A}^+$ .

A substitution is *right proper* if there exists a letter  $a \in \mathcal{A}$  such that the rightmost letter of  $\phi(b)$  is  $a$  for all  $b \in \mathcal{A}$ . That is,  $\phi(b) = w_b a$  for some  $w_b \in \mathcal{A}^+$ .

A substitution is *fully proper*<sup>1</sup> if it is both left and right proper.

A substitution is *pre-left proper* if some power of the substitution is left proper. Similarly for *pre-right proper* and *pre-fully proper*.

<sup>1</sup>We will often abbreviate this to just *proper*.

The following algorithm produces what we call the *pre-left properisation* of a substitution, so-called because there exists a finite power of the new substitution which is left proper. As per usual  $k$  will be the one from the definition of the fixed letter  $f$ .

Note that if  $v$  is a return word to the fixed letter  $a$ , then  $va \in \mathcal{L}$  and so  $\phi^k(va)$  must also be admitted by  $\phi$ . But  $\phi^k(va) = \phi^k(v)\phi^k(a)$ , and both of  $\phi^k(v)$  and  $\phi^k(a)$  begin with the fixed letter  $a$ , hence  $\phi^k(v)$  is an exact composition of return words to  $a$ . So, if we apply  $\phi^k$  to a return word, then the result is a composition of return words. We will denote by  $\psi$  this newly constructed substitution rule on the new alphabet  $\mathcal{R}$  of return words.

---

**Algorithm 7** Pre-left properisation

---

```

1: function preprop
2:   rwords = returnwords(f)
3:    $\psi$  = empty substitution with alphabet size being the cardinality of rwords
4:   for all words  $w$  in rwords do {
5:     temp =  $\phi^k(w)$ 
6:     decomposition = decompose temp into return words  $w_{i_1} \dots w_{i_m}$ 
7:      $\psi(w)$  = decomposition
8:   }
9:   output  $\psi$ 

```

---

This algorithm gives us a new substitution on possibly more letters than with what we began, and we may take a power of this substitution to get a left proper one. That such a power exists is clear. Indeed, every return word  $v \in \mathcal{R}$  begins with the fixed letter  $a$  and, by primitivity,  $\phi^i(a)$  contains at least two copies of the letter  $a$  for large enough  $i$ , so  $\phi^i(a) = v_0 a u$  for some return word  $v_0 \in \mathcal{R}$  and some other word  $u$ . But then  $\psi^i(v)$  must begin with  $v_0$ . It follows that  $\psi^i$  is a left-proper substitution with leftmost letter  $v_0$ .

We may also form an equivalent fully proper substitution on  $\mathcal{R}$  by composing  $\psi^i$  with its *right conjugate*. The right conjugate  $\phi^{(R)}$  of a left proper substitution  $\phi$  is given by setting  $\phi^{(R)}(b) = w_b a$  where  $a$  is the fixed letter such that  $\phi(b) = a w_b$  for all  $b \in \mathcal{A}$ . The right conjugate is a right proper substitution, and the composition of a left proper and right proper substitution is both left and right proper, hence fully proper. It is easy to show (See [14]) that  $X_{\phi \circ \phi^{(R)}}$  and  $X_\phi$  are topologically conjugate subshifts. In fact, a word is admitted by  $\phi \circ \phi^{(R)}$  if and only if it is admitted by  $\phi$ , so  $X_{\phi \circ \phi^{(R)}}$  and  $X_\phi$  are equal. Hence,  $\check{H}^1(\Omega_{\phi \circ \phi^{(R)}}) \cong \check{H}^1(\Omega_\phi)$ .

We make use of the following which has been paraphrased from results appearing in the work of Durand, Host and Skau in [13].

**Proposition 3.5.** *Let  $\phi$  be a primitive substitution on  $\mathcal{A}$  and let  $\psi$  be the pre-left properisation of  $\phi$ . The tiling space  $\Omega_\psi$  is homeomorphic to  $\Omega_\phi$ .*

Hence we get the corollary

**Corollary 3.6.**

$$\check{H}^1(\Omega_\psi) \cong \check{H}^1(\Omega_\phi)$$

As  $\psi^i$  is left proper,  $\check{H}^1(\Omega_{\psi^i}) \cong \varinjlim M_{\psi^i}^T \cong \varinjlim (M_\psi^i)^T \cong \varinjlim M_\psi^T$ . Hence  $\check{H}^1(\Omega_\phi) \cong \varinjlim M_\psi^T$ .

Grout outputs the pre-left properisation  $\psi$ , the left properisation  $\psi^i$ , the full properisation  $\psi^i \circ (\psi^i)^{(R)}$ , and owing to the above, Grout also outputs the matrix  $M_\psi^T$  in the cohomology section.

Pre Left Properisation :	Left Properisation :	Full Properisation :
$a \mapsto b$	$a \mapsto b$	$a \mapsto ba$
$b \mapsto ba$	$b \mapsto ba$	$b \mapsto bba$

FIGURE 7. The properisation results of the Fibonacci substitution

**Remark 3.7.** If the substitution is already proper, the properisation algorithm may return a different proper version of this substitution. This may seem like it is a feature which has no use, but by iterating this process, we find that the sequence of substitutions is eventually periodic, first proved by Durand [12]. It was therefore decided to leave this feature intact, in order to study such sequences of properisations.

Barge-Diamond :	Properisation :	Anderson-Putnam :
$\lim M^T$	$\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}$
Cohomology Rank : 2		

FIGURE 8. The cohomology results of the Fibonacci substitution

#### 4. FURTHER RESULTS

In this section we provide the  $\text{\LaTeX}$  output of cohomological calculations from Grout for a selection of both well-known and not so well-known substitutions appearing in the literature.

- (1) The *Thue-Morse substitution* is one of the most well-studied substitutions in symbolic dynamics [23], possibly only superseded by the Fibonacci substitution in the attention it has received. It would be remiss to include a list of example outputs which did not include the results for this substitution.
- (2) The *Tribonacci substitution* is an example of a unimodular irreducible Pisot substitution, first described by Rauzy in his seminal paper [24] introducing the so-called Rauzy fractals, and still actively studied for its interest to symbolic dynamicists and fractal geometers.
- (3) The *Disconnected Subcomplex substitution* was first described by Barge and Diamond [4] as an example of a substitution whose Barge-diamond complex has a disconnected subcomplex of edges labelled by two letter words. This is reflected in the cohomology calculation via the Barge-Diamond complex, where a non-trivial quotient appears according to the formula for the cohomology described in Corollary 3.2.
- (4) The Fibonacci and Tribonacci substitutions are the first and second in an infinite family of primitive recognisable substitutions which take the form

$$\begin{aligned} a_j &\mapsto a_1 a_{j+1}, & \text{if } 1 \leq j < n \\ a_n &\mapsto a_1 \end{aligned}$$

for an alphabet  $\{a_1, \dots, a_n\}$  on  $n$  letters. One might call these substitutions the *n-ibonacci substitutions*. We have chosen to show the output for the Hexibonacci substitution where  $n = 6$ , as the associated Barge-Diamond complex is particularly pleasing.

#### 4.1. Thue-Morse.

$$\begin{array}{lcl} a & \mapsto & ab \\ b & \mapsto & ba \end{array}$$



Substitution Matrix :

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Full Properisation :

$$\begin{array}{lcl} a & \mapsto & cacbacab \\ b & \mapsto & cbabcacbabcbacab \\ c & \mapsto & cbabcacbacabcacbabcbacab \end{array}$$

Barge Diamond Cohomology Group :  $\varinjlim M^T \oplus \mathbb{Z}^1$

Properisation Cohomology Matrix :

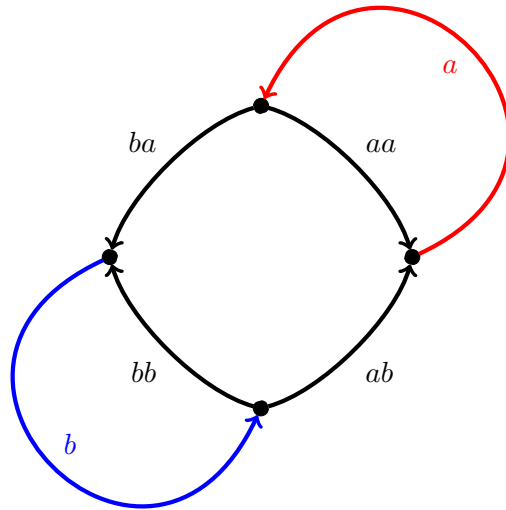
$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Anderson-Putnman Cohomology Matrix :

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

Cohomology Rank : 2

**Barge-Diamond Complex**



#### 4.2. Tribonacci.

$a \mapsto ab$   
 $b \mapsto ac$   
 $c \mapsto a$



Substitution Matrix :

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Fixed Letter : a

Return Words : a, ab, ac

Recognisable: Yes

Full Properisation :

$a \mapsto bc$   
 $b \mapsto babc$   
 $c \mapsto bbc$

Barge Diamond Cohomology Group :  $\varinjlim M^T$

Properisation Cohomology Matrix :

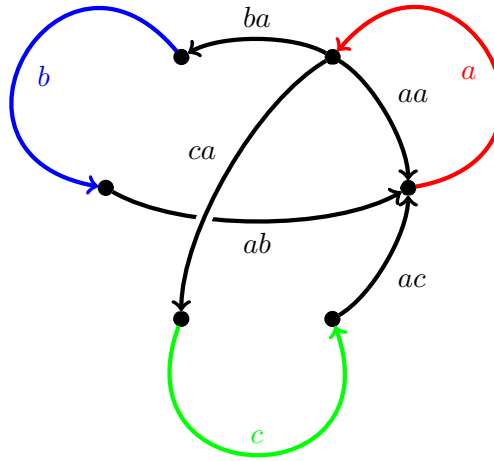
$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Anderson-Putnman Cohomology Matrix :

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Cohomology Rank : 3

**Barge-Diamond Complex**



#### 4.3. Disconnected Subcomplex.

$$\begin{aligned} a &\mapsto abcd a \\ b &\mapsto ab \\ c &\mapsto cdb c \\ d &\mapsto db \end{aligned}$$



Substitution Matrix :

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Fixed Letter : a



Return Words : a, ab, abcd, abcdcbdb

Recognisable: Yes

Full Properisation :

$a \mapsto cacad$   
 $b \mapsto cacabcad$   
 $c \mapsto cacaddbcad$   
 $d \mapsto cacaddbcaddbcabcad$

Barge Diamond Cohomology Group :  $\varinjlim M^T / \mathbb{Z}^1$

Properisation Cohomology Matrix :

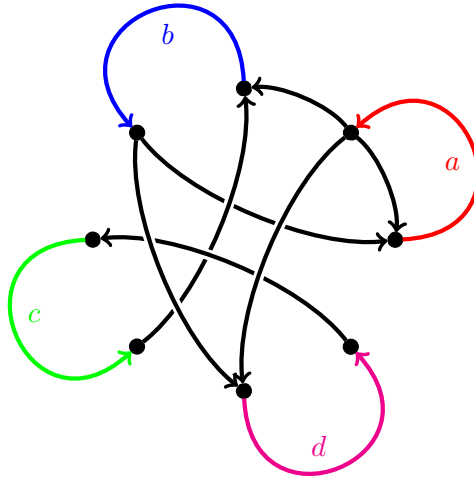
$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

Anderson-Putnman Cohomology Matrix :

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Cohomology Rank : 3

**Barge-Diamond Complex**



#### 4.4. Hexibonacci.

$a \mapsto ab$   
 $b \mapsto ac$   
 $c \mapsto ad$   
 $d \mapsto ae$   
 $e \mapsto af$   
 $f \mapsto a$



Substitution Matrix :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Fixed Letter : a

Return Words : a, ab, ac, ad, ae, af

Recognisable: Yes

Full Properisation :

$a \mapsto bc$   
 $b \mapsto bdbc$   
 $c \mapsto bebc$   
 $d \mapsto bfbcb$   
 $e \mapsto babcb$   
 $f \mapsto bbc$

Barge Diamond Cohomology Group :  $\varinjlim M^T$

Properisation Cohomology Matrix :

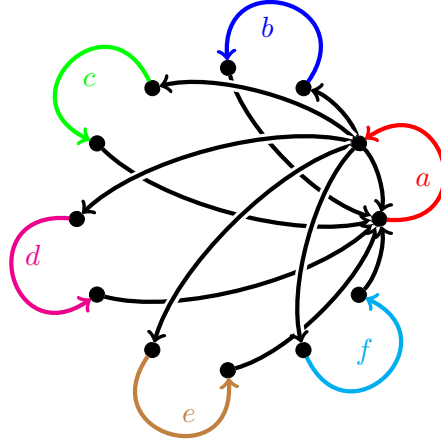
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Anderson-Putnman Cohomology Matrix :

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Cohomology Rank : 6

### Barge-Diamond Complex



### REFERENCES

- [1] Qt version 5.4.1, 2015. <http://www.qt.io>.
- [2] J. E. Anderson and I. F. Putnam. Topological invariants for substitution tilings and their associated  $C^*$ -algebras. *Ergodic Theory Dynam. Systems*, 18(3):509–537, 1998.
- [3] M. Baake and U. Grimm. *Aperiodic order. Vol. 1*, volume 149 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2013. A mathematical invitation, With a foreword by Roger Penrose.
- [4] M. Barge and B. Diamond. Cohomology in one-dimensional substitution tiling spaces. *Proc. Amer. Math. Soc.*, 136(6):2183–2191, 2008.
- [5] J. Bellissard, D. J. L. Herrmann, and M. Zarrouati. Hulls of aperiodic solids and gap labeling theorems. In *Directions in mathematical quasicrystals*, volume 13 of *CRM Monogr. Ser.*, pages 207–258. Amer. Math. Soc., Providence, RI, 2000.
- [6] R. Berger. *The Undecidability of the Domino Problem*. Memoirs ; No 1/66. American Mathematical Society, 1966.
- [7] V. Berthé and A. Siegel. Tilings associated with beta-numeration and substitutions. *Integers*, 5(3):A2, 46, 2005.
- [8] R. Bott and L. Tu. *Differential Forms in Algebraic Topology*. Graduate Texts in Mathematics. Springer New York, 1982.
- [9] A. Clark and J. Hunton. Tiling spaces, codimension one attractors and shape. *New York J. Math.*, 18:765–796, 2012.
- [10] K. Culik, II. The decidability of  $v$ -local catenativity and of other properties of D0L systems. *Information Processing Lett.*, 7(1):33–35, 1978.
- [11] D. Damanik and D. Lenz. Substitution dynamical systems: characterization of linear repetitivity and applications. *J. Math. Anal. Appl.*, 321(2):766–780, 2006.
- [12] F. Durand. A characterization of substitutive sequences using return words. *Discrete Math.*, 179(1-3):89–101, 1998.

- [13] F. Durand, B. Host, and C. Skau. Substitutional dynamical systems, bratteli diagrams and dimension groups. *Ergodic Theory and Dynamical Systems*, 19:953–993, 8 1999.
- [14] F. Durand and J. Leroy. S-adic conjecture and bratteli diagrams. *Comptes Rendus Mathématique*, 350(21-22):979 – 983, 2012.
- [15] A. Ehrenfeucht and G. Rozenberg. On simplifications of PDOL systems. In *Proceedings of a Conference on Theoretical Computer Science (Univ. Waterloo, Waterloo, Ont., 1977)*, pages 81–87. Comput. Sci. Dept., Univ. Waterloo, Waterloo, Ont., 1978.
- [16] F. Gähler and G. R. Maloney. Cohomology of one-dimensional mixed substitution tiling spaces. *Topology Appl.*, 160(5):703–719, 2013.
- [17] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.
- [18] T. Harju and M. Linna. On the periodicity of morphisms on free monoids. *Informatique théorique et applications*, 20(1):47–54, 1986.
- [19] A. Julien. Complexity as a homeomorphism invariant for tiling spaces. *Preprint*, arXiv:1212.1320, 2012.
- [20] A. Lagae. *Wang Tiles in Computer Graphics*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2009.
- [21] B. Mossé. Puissances de mots et reconnaissabilité des points fixes d’une substitution. *Theoret. Comput. Sci.*, 99(2):327–334, 1992.
- [22] R. Penrose. Pentaplexity a class of non-periodic tilings of the plane. *The Mathematical Intelligencer*, 2(1):32–37, 1979.
- [23] E. Prouhet. Mémoire sur quelques relations entre les puissances des nombres. *C. R. Acad. Sci. Paris Sér.*, 1(33):225, 1851.
- [24] G. Rauzy. Nombres algébriques et substitutions. *Bull. Soc. Math. France*, 110(2):147–178, 1982.
- [25] L. Sadun. *Topology of tiling spaces*, volume 46 of *University Lecture Series*. American Mathematical Society, Providence, RI, 2008.
- [26] D. Shechtman, I. Blech, D. Gratias, and J. W. Cahn. Metallic phase with long-range orientational order and no translational symmetry. *Phys. Rev. Lett.*, 53:1951–1953, Nov 1984.
- [27] R. Twarock. A tiling approach to virus capsid assembly explaining a structural puzzle in virology. *Journal of Theoretical Biology*, 226(4):477–482, 2004.